DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
THE UNIVERSITY OF TEXAS AT ARLINGTON

ARCHITECTURAL DESIGN SPECIFICATION
CSE 4317: SENIOR DESIGN II
FALL 2019

NEBULOO
FALCON

ALEJANDRO WAUMANN
JUAN BARRAGAN
LUIS GONZALEZ
SUNIL NIROULA

Revision History

Revision	Date	Author(s)	Description
0.1	08.05.2019	AW,JB,LG,HT,SN	document creation
0.2	10.25.2019	LG	Added more info and team logo
0.3	12.06.2019	LG	Final version of doc created

Contents

List of Figures

List of Tables

1 Introduction

Falcon is an artificial intelligence (AI) system that controls a anti-aircraft ground-mounted turret. It will be used to demo the use of Dream Shader, a simulation engine that can be used to train an AI system and then verify and validate it through various test scenarios. Those will be the primary components of the system, along with a controller. The controller will take in data from the AI agent layer and the simulation engine layer to decide what actions the turrent should take. This can be information about the environment being simulated that the system is expected to receive in regular field tests as well as action suggestions from the AI agent.

2 System Overview

The simulation engine layer will feed the AI agent layer a framebuffer. It will also feed the controller layer information about the state of the simulation and settings that the user may have set that should influence the move evaluation. The AI agent will give the simulation engine a modified framebuffer since it will act as a filter that add bounding boxes and label to objects that it recognizes. It will also send its move suggestion to the controller layer. The controller layer will only send data to the simulation engine layer to alter the state of the simulation.

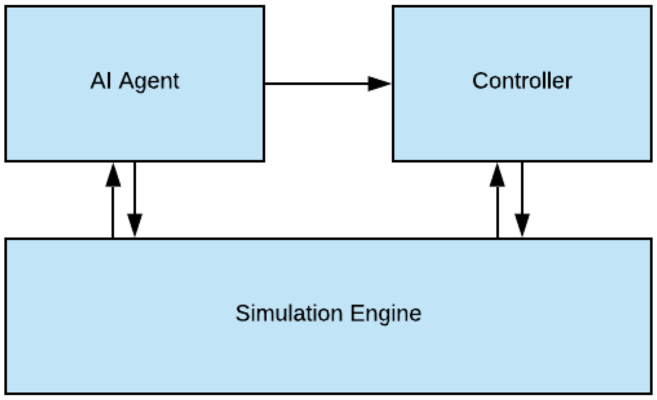

Figure 1: A simple architectural layer diagram

2.1 AI Agent Layer Description

The AI agent layer will consist of three subsystems. There will be an object detection subsystem, an object recognition subsystem, and an object tracking subsystem. This layer will receive a framebuffer and alter it to change the simulation state and send its move evaluation to the controller layer.

2.2 Controller Layer Description

The controller layer will consists of two subsystems. There will be a state subsystem that contains data related to the state of the layer and a move evaluator subsystem that will decide what action the turret should take. The move evaluator will make decisions based on the state subsystem. Any module that would like to influence the control of the turret will need to modify the state of the controller.

2.3 Simulation Engine Layer Description

The simulation engine layer will consist of several subsystems. It will contain the state of the simulation, a rendering engine, a model loader, an event system, a windowing system, and a camera implementation. The state of the simulation will contain all the data related to the environment and everything in it. The rendering engine will be used to render the environment using vulkan. The model loader will simply load model data that is in the gltf2 format. This is the only format that will be supported on initial release. The event system will be responsible for taking care of user input and notifying the proper subsystems of it. The windowing system just allows the developer to create and manage windows on the Linux operating system. The camera implementation will simulate the view of a camera mounted on the turret. Other camera implementations may be used to simulate other views. These subsystems will mainly interact by modifying the state of the simulation.

3 Subsystem Definitions & Data Flow

The controller will use the renderer to create models and set their transformations. The controller will then use the dedicated input system to process the input and take action. The rendering engine will provide an image that the controller can take and hand off to the AI system. The AI system will specify the bounding box coordinates that the rendering engine should draw. Then, the controller will hand off the rectangle coordinates to the rendering engine. Finally, the rendering engine will draw the rectangle.

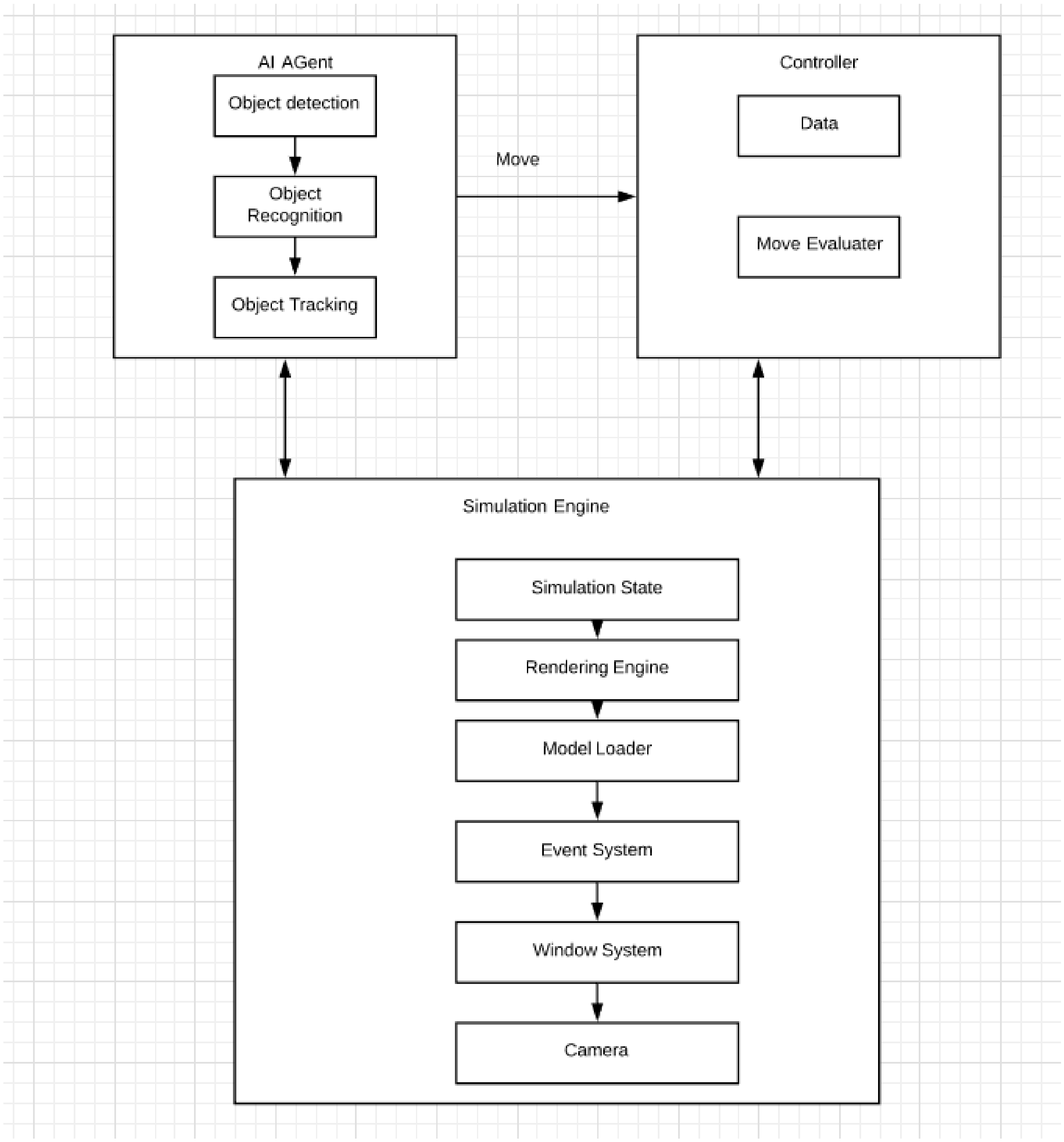

Figure 2: A simple data flow diagram

4 AI Agent Layer Subsystems

This layer is particularly responsible for object detection, object recognition and object tracking. The first subsystem under AI agent is going to be object detection which will use some algorithm to detect the object through the video feed that is provided to the algorithm.

4.1 Object Detection Subsystem

This subsystem uses YOLO v3 algorithm to detect the objects in an image. This interacts with the object recognition subsystem under the AI agent system. This subsystem will take the video feed from the camera and give the input as an image to the algorithm and the algorithm will detect if the object we are looking for is present in the image or not. This is a continuous process as in our project we are looking for a plane that is flying so it has to check all the possible feeds and find the object and shoot it down.

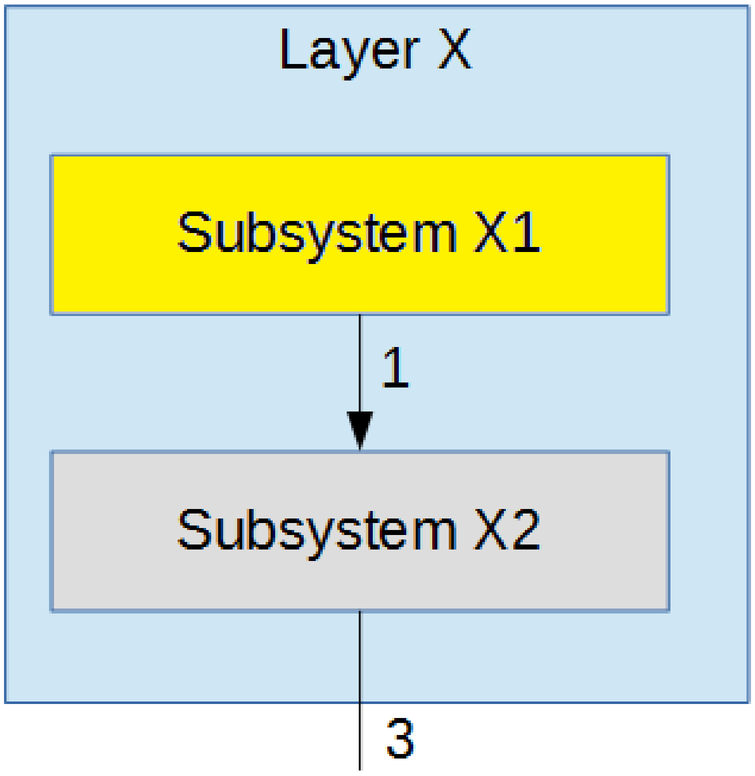

Figure 3: Example subsystem description diagram

4.1.1 Assumptions

We assume some of the things before using the YOLOv3 algorithm. One of them is bounding box prediction. Objectness score is predicted using the logistic regression. It is 1 if the bounding box prior overlaps a ground truth object by more than one bounding box prior. Only one bounding box is assigned for each ground truth object. The second one is class prediction. Independent logistic classifiers is used and binary cross entropy is used. Because there may be overlapping labels for multilabel classification.

4.1.2 Responsibilities

The basic responsibility of this subsystem is to detect object that we are looking for. In our case it has to detect planes and other objects. Our primary focus is to detect the enemy planes and shoot them down. The algorithm must detect the objects inside the certain image. It should be very accurate and fast. This subsystem will then send the detected objects as inputs to the object recognition subsystem where the objects will be recognized based on their attributes.

4.1.3 Subsystem Interfaces

Table 2: Subsystem interfaces

ID	Description	Inputs	Outputs
1	A filter that adds bounding boxes and labels to a framebuffer.	framebuffer	framebuffer

4.2 Object Recognition Subsystem

After our object detection algorithm scans through an input video feed or image and localizes all objects of interest, it passes control to the object recognition subsystem to handle the next important task â classifying detected objects into classes that the application cares about. Irrelevant objects will be filtered out to decrease the noise and to allow objects of significant interest to stand out.

4.2.1 Assumptions

Since the output of object detection is the input of this subsystem, we assume that the object detection phase uncovers most, if not all, objects of interest. Object recognition will only operate on the pool of objects that have been detected. Therefore, any missing object will not be correctly classified and labeled.

4.2.2 Responsibilities

The main responsibility of this subsystem is to predict the type of classes of certain objects of interest, which include airplanes, missiles, etc. Depending on which category a given object belongs to, it might be dropped from consideration. It is important to point out that not only does this subsystem tells the next module (object tracking) to not track certain objects, but it should also give more weight to high value objects that should be carefully tracked. A rather contrived example, but hopefully illustrative, is a commercial airliner and a military jet. While both should be tracked, extra weight should naturally be given to the military jet.

4.3 Object Tracking Subsystem

Object tracking is another crucial part of this project and it falls right under the AI Agent Subsystem. This subsystem will take on the data provided by the Object Recognition Subsystem and continue on with the task of continuously tracking the desired object recognized by the previous subsystem.

4.3.1 Assumptions

In order for this subsystem to perform its intended task correctly, we will assume that the data that was passed on to this subsystem from the other members within the AI Agent Layer is correct so that we actually end up tracking the right objects. Also, we will have to assume that whatever tracker we actually end up implementing out of the ones available to us from OpenCV will be the best for tracking the objects we need and within an acceptable amount of time and computing effort.

4.3.2 Responsibilities

Object tracking will continue to be done using 1 of 8 different trackers available in OpenCV 3.4.1(Version is subject to change) that may suit our needs the best: MIL, TLD, KCF, CSTR, MOSSE, GOTURN, BOOSTING and MEDIANFLOW. No matter the tracker we end up using, the goal will continue to be to track objects identified by the previous subsystem within an acceptable time frame(to be decided) and pass on the results onto the next layer for further processing.

4.3.3 Subsystem Interfaces

Table 3: Subsystem interfaces

ID	Description	Inputs	Outputs
1	Take in data from previous subsystem of identified object, and continue with the tracking of desired object; relay location of tracked object to the next layer.	Object to be tracked	Continuous feed of object location

5 Controller Layer Subsystems

This layer is responsible for making decisions based on the information given. The subsystems in the Controller layer are Data and Move Evaluator.

5.1 Data Subsystem

This subsystem collects the ÃmoveÃ information given to it by the AI Agent and any other information given to it from the Simulated Engine.

5.1.1 Assumptions

We assume that the information given to this subsystem, from the other layers, is correct.We also assume that the data being collected will be to influence the outcome of every decision made by the program.

5.1.2 Responsibilities

The basic responsibility of this subsystem is to collect information given to it by the other subsystems and store it for usage.

5.2 Move Evaluator Subsystem

This subsystem takes the data that has been collected and decides the appropriate outcome for each individual situation.

5.2.1 Assumptions

We assume that the information given to this subsystem, from the other layers, is correct.We also assume that the move evaluator will be able to make the correct decision based on the data it has been given.

5.2.2 Responsibilities

The basic responsibility of this subsystem is to use the data that has been collected, and make decisions for the program, without error.

6 Simulation Engine Layer Subsystems

6.1 Simulation State Subsystem

The simulation state subsystem holds all of the data related to the state of the simulation. This controls how the renderer will draw things, how the controller layer will make decisions, and many other things that rely on the state of the simulation.

6.1.1 Assumptions

This subsystem assumes that most interation in the system will occur by modifying this state. Subsystems and other layers that need specific data will get it by checking this state.

6.1.2 Responsibilities

This subsystem has a set of functions that allows the user to modify and read the state of various aspects of the simulation and the environment that was generated.

6.1.3 Subsystem Interfaces

Table 4: Subsystem interfaces

ID	Description	Inputs	Outputs
#01	State of various aspects of the simulation.	State	State

6.2 Renderer Subsystem

The renderer subsystem will be tasked with rendering the scene based on the state of the simulation.

6.2.1 Assumptions

The subsystem assumes that the state will be modified properly to render to correct scene.

6.2.2 Responsibilities

This subsystem is responsible for providing the user with function to render the scene.

6.3 Model Loader Subsystem

The model loader will load models into the scene.

6.3.1 Assumptions

This subsystem assumes that all models will be in the gltf2 format.

6.3.2 Responsibilities

This subsystem is responsible for reading in a gltf2 model and making the model data available to the developer.

6.3.3 Subsystem Interfaces

Table 5: Subsystem interfaces

ID	Description	Inputs	Outputs
#01	Loads a gltf2 model into the scene	gltf2 model	model data

6.4 Event System Subsystem

The event system subsystem will take care of the user input from the mouse and keyboard as well as other window events such as window resizing, closing the window, etc.

6.4.1 Assumptions

This subsystem assumes that the OS being used is Linux and that Vulkan is being used to render the scene.

6.4.2 Responsibilities

This subsystem is responsible for allowing the developer to define functions that should be run upon specific input events.

6.4.3 Subsystem Interfaces

Table 6: Subsystem interfaces

ID	Description	Inputs	Outputs
#01	A user-defined function is called when specific user input is received.	user input	call function
#02	A user-defined function is called when specific window event occurs.	window event	call function

6.5 Window System Subsystem

The window subsystem provides a context for the vulkan to draw to. It allows the developer to create and manage multiple windows to use for different views into the simulation.

6.5.1 Assumptions

This subsystem assumes that Vulkan is being used to render the scene and that the Linux OS is being used.

6.5.2 Responsibilities

This subsystem is primarily responsible for providing window events and the rendering context.

6.6 Camera Subsystem

The camera subsystem provides various views into the simulation. In our demo, it will be used to simulate a camera mounted onto the turret.

6.6.1 Assumptions

This assumes that a camera is being simulated. This also assumes that the camera movement will be controlled by the controller later via the simulation state subsystem.

6.6.2 Responsibilities

This subsystem is responsible for allowing the developer to control the camera movements and show different views into the simulation.

6.6.3 Subsystem Interfaces

Table 7: Subsystem interfaces

ID	Description	Inputs	Outputs
#01	Controller layer allows for movement of the camera view.	Movement Input	camera movement

References